Data Dependent Sparing to Manage Better-Than-Bad Blocks

Rakan Maddah, Sangyeun Cho, and Rami Melhem
Computer Science Department, University of Pittsburgh
{rmaddah,cho,melhem}@cs.pitt.edu

Abstract—We forecast that proper handling of unreliable storage blocks (e.g., "bad block management" in solid-state drives) will remain critical for future systems built with advanced and emerging memory technologies. This paper argues that the conventional block retirement and sparing approach—a block is retired as soon as it shows faulty behavior—is overly conservative and inefficient. We observe that it is highly unlikely that all faulty bits in a storage block manifest errors. Consequently, we propose *data dependent sparing*, a relaxed block retirement and sparing approach that recycles faulty storage blocks. At small management cost and with less than 1% sparing, data dependent sparing achieves the same lifetime as the conventional approach with 20% sparing.

Index Terms—Sparing, phase-change memory (PCM), flash memory, solid-state drive (SSD), stuck-at faults.

1. Introduction

Bad block management is a vital technique for memories subject to relatively low write endurance. For example, the NAND flash memory medium in modern solid-state drives (SSDs) has a write endurance of only 10^3 to 10^5 [6]. While SSDs perform aggressive *wear leveling* to spread writes to the entire flash memory capacity, careful bad block management is required to cope with the unreliable blocks and preserve the dependability and performance of the storage device. The common practice is to permanently retire a bad block, once manifested, and replace it with a good spare block [7], [8]. In flash SSDs, over-provisioning of as large as 20% is typical in server products.

Effective bad block management is expected to remain critical for emerging memories like phase change memory (PCM), when deployed in main memory and secondary storage alike [12]. Initial measurements indicate that PCM cells can endure 10^6 or more write cycles [9]. This rating is certainly better than the 10^3 to 10^5 write endurance of recent flash memory, but not by far. Realistic PCM based SSD prototypes demonstrated a superior performance potential to that of flash SSDs [1], [5]. Accordingly, under demanding applications PCM SSDs will have to absorb a larger write volume during the lifetime than current NAND flash SSDs.

In NAND flash memory, repeated program/erase cycles damage the nitride layer of a cell, causing charges to be trapped in the dielectric. This mechanism results in a permanent shift in the cell state. Writing to PCM cells requires repeated heating and cooling (expansion and contraction) of the chalcogenide material, leading to the detachment of the chalcogenide from the heating elements. This mechanism also results in a permanent shift in the cell state. Accordingly, NAND flash as well as PCM cells exhibit a "stuck-at" fault model [4], [7], [12], [13]. That is, when a cell fails, it gets stuck at either 0 or 1, and can still be read but not reprogrammed. In this case, the manifestation of errors is *data dependent*; an error occurs only when a different bit value is written to a faulty cell than what it is stuck at. As an example, consider a block with twenty stuck-at faults. Writing new data to the block may result in only nine errors—when eleven remaining faulty cells were written bit values identical to their stuck-at values.

Given an error correction scheme capable of masking N errors, a write operation fails if $N+1$ or more errors are manifested. Accordingly, the probability of failure for a block with F faults is $P(F,N) = \sum_{f=N+1}^{F} \binom{F}{f} (\frac{1}{2})^F$ for $F > N$. Fig. 1 plots the probability of failure on a block write as a function of the number of stuck-at faults in the block. The plot assumes that the block is protected by an error correction mechanism capable of fixing up to 20 errors. It shows that a block write fails rarely even if there are more than 20 faults. For example, when a block has 29 faults, the probability of block write failure, i.e., at least 21 errors are manifested, is only ∼1%. Clearly, retiring a block immediately when it becomes faulty (having more than 20 faults given the capability of the error correction mechanism) is overly conservative and fails to squeeze more lifetime from faulty, yet "better-than-bad" blocks.

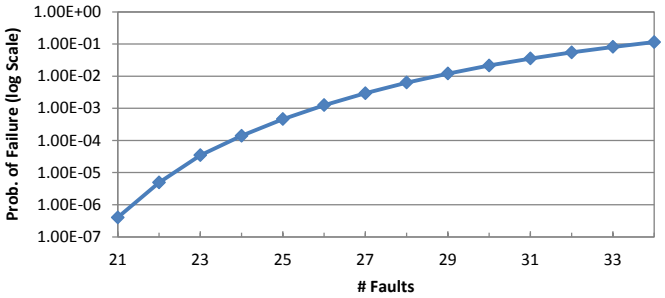

Fig. 1. Block write failure probability vs. # of faults within a 4KB storage block, when an error correction mechanism covers up to 20 errors.

This paper proposes *data dependent sparing*, a new physical block sparing scheme that delays the retirement of a faulty block when the memory exhibits the stuck-at fault model. Specifically, we do not immediately retire a block on an unsuccessful write; instead, we borrow a spare block temporarily to complete the write operation. When we perform a later write operation on the original (faulty) block, the operation will succeed with a high probability (i.e., new data are different from old data). Essentially, we classify storage blocks into "good", "better-than-bad" and "bad" (retired) and utilize both good and better-than-bad blocks on writes. A better-than-bad block is reclassified as bad when it suffers frequent failures.

To manage better-than-bad blocks, we need to keep track of their "goodness", which may be achieved with a counter per block that records past failures. Alternatively, one could employ storage-efficient global data structures like Bloom filters to bookkeep failing blocks. Our quantitative result shows that data dependent sparing substantially improves the lifetime of a storage device, justifying the small management overhead. Lifetime improvement as compared to a conventional scheme

Manuscript submitted: 30-Apr-2012. Manuscript accepted: 10-Jun-2012. Final manuscript received: 13-Jun-2012.
This work is supported in part by NSF grants CCF-1064976, CCF-1059283, and CNS-1012070.

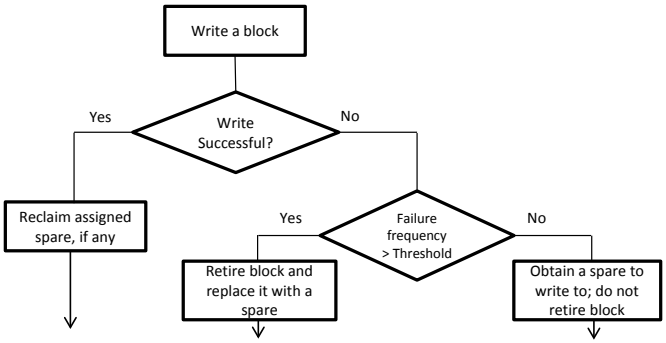

Fig. 2. Flow of execution in data dependent sparing.

is as large as 24.2% at 20% sparing. In turn, data dependent sparing achieves an equivalent lifetime as the conventional scheme with significantly fewer spares.

2. DATA DEPENDENT SPARING

Data dependent sparing builds on the observation that faulty blocks[1] have a low probability of actual write failure. This section describes in detail the idea of data dependent sparing, design trade-offs and its overheads.

2.1. Conceptual Design

When an attempt to write to a block fails, i.e., there are more errors than can be corrected by the error correction mechanism, a spare block replaces the original block.[2] In conventional bad block management, referred to as "static sparing" in this paper, this block is labeled "bad" and is discarded at once [7], [8], [10]. In data dependent sparing, the original block is not immediately retired because a later write to the same block with different data will likely succeed.

When a later write to the block succeeds, the initially assigned spare is reclaimed to the spare pool, essentially increasing the write volume of the device without help from a spare. After more write cycles are applied, however, the block will wear out further and become unreliable. Fig. 1 shows that the probability of write failure increases with the number of faults in the block. Hence, once a block is deemed to have a high probability of producing write failure, it is retired to prevent repeated writes and associated overheads. Data dependent sparing introduces a *failure frequency threshold* to determine when a block is retired. Data dependent sparing's flow of execution is depicted in Fig. 2.

Data dependent sparing is expected to significantly prolong the lifetime of a storage device. By continuously involving better-than-bad blocks in data writes and dynamically allocating spares to temporarily accommodate writes that failed, data dependent sparing utilizes and manages sparse spares more efficiently than conventional bad block management. Section 3 quantitatively studies the benefits of data dependent sparing.

It is worthwhile to note a recently proposed scheme called Pay-As-You-Go (PAYG) [11]. It reduce the storage for error correction through allocating error correction entries in proportion to the number of hard-faults in a block. PAYG is orthogonal to our scheme. Specifically, it delays the failure of blocks but do not reuse bad blocks. Moreover, it does not exploit the data-dependent nature of failures.

1. A storage block is *faulty* when it has more stuck-at faults than the provided error correction mechanism can always cover.

2. A *block* is the unit of write and error correction. It is not to be confused with NAND flash memory "block" (multiple of "pages").

2.2. Design Trade-Offs

There are several interesting design decisions to be made when realizing data dependent sparing. The first has to do with *how spares are allocated upon a write failure*. Two strategies are feasible: (1) **Temp-sparing**: a healthy spare block temporarily substitutes the failing better-than-bad block and completes the write request; and (2) **Role-exchange**: a spare block permanently replaces the failing block and the failing block is added to the pool of spares. Temp-sparing has the advantage of reducing the wear of spares because the spares will be written infrequently compared with regular blocks. This strategy maximizes the chances of finding a healthy spare quickly when needed. Role-exchange has the advantage of spreading writes across the entire capacity of the device including spares. Hence, regular and spare blocks will wear at the same rate. It may incur multiple writes on spares more frequently than temp-sparing.

The second design choice is about how to map a logical block (that failed) to a new physical block. If the temp-sparing strategy is applied, a small table could be implemented to store pointers to spare blocks. The number of entries in the table is equal to the number of provided spares. If the role-exchange strategy is applied, then this requires address remapping , e.g., updating the address remapping table in SSD, so that read and write accesses are mapped to the new block. These two design choices are similar to Micron's *Skip Block Method* and *Reserve Block Method* [8].

Finally, we need a mechanism to determine when a better-than-bad block retires. This mechanism kicks in when a block write fails and is queried about the block's history of failures. If the block is failing more frequently than a *failure frequency threshold*, it is retired. A straightforward strategy to track past failure history is to associate a counter with each (better-than-bad) block that records the number of failures. In another strategy, one could employ a global data structure that approximates individual counters, like a counting Bloom filter. If the number of better-than-bad blocks is expected to be small in a device during its life (e.g., the device steers writes to known better-than-bad blocks to exhaust them first), this strategy could result in smaller bookkeeping space overheads than the first strategy.

2.3. Overheads

In data dependent sparing, a single block write operation may incur a series of writes before completion due to error occurrences, adding to the performance cost of a write. However, this cost is expected to be very small because erroneous writes are extremely rare. If the temp-sparing strategy is applied, only one extra write is expected as this strategy preserves the health of spares. If the role-exchange strategy is applied, multiple writes could occur as the spare wears at the same level of other blocks. However, multiple writes will occur with a low probability. For example, consider a failure frequency threshold of 1/100. The probability of a write operation failing twice in a row for a given data pattern is at most 1/10,000.

The second source of overheads in data dependent sparing is the data structure to bookkeep history of errors. If data dependent sparing introduces a one-byte counter per 4KB block, this overhead corresponds to 1/4,096 = 0.24%. The overhead can be made smaller if we allocate counters on demand (i.e., no counter for a healthy block) or use an approximation data structure like Bloom filter.

Lastly, a major storage capacity overhead—of up to 20%—comes from over-provisioned capacity (the amount of spares). Spares are consumed as bad blocks occur, and hence, a storage device must provision sufficient spares to guarantee a target lifetime. Because data dependent sparing increases the write volume each storage block successfully absorbs, it effectively

delays the wearing of available blocks and increases a storage device's lifetime. In turn, compared with static sparing, data dependent sparing reduces the overhead due to over-provisioning given a target lifetime.

3. Evaluation

This section evaluates the performance of data dependent sparing, focusing on the relative advantage of data dependent sparing compared with static sparing in terms of lifetime improvement and reduction in required over-provisioning given a lifetime target.

We resort to Monte Carlo simulation in evaluation. Since detailed simulation of a large storage capacity is impractical, we simulate 2,000 storage blocks of 4KB and derive results statistically. Cells in the storage blocks have a write endurance following a Gaussian distribution. We choose two configurations. Cells in the "Flash" configuration have a mean of 8.27×10^5 with a standard deviation of 2.48×10^5 [2]. "PCM" cells show a mean of 10^8 and a standard deviation of 25×10^6 [12]. Overall, our methodology is similar to the one in [12].

We assume that an efficient wear leveling scheme distributes writes evenly across the available storage blocks. Each storage block is protected with an error correction code. We use a BCH code built over a Galois field of size 2^{16} [3], capable of correcting n errors (BCH-n) where n is a parameter. We keep track of the number of faults within a block. Once the number of faults gets above BCH capability, the success of a write operation is determined based on the probability of failure as presented in Fig. 1.

Because the primary concern in this paper is to cover hard faults, we assume that the BCH capability is dedicated to masking hard faults. In the case of NAND flash memory, there are other important transient faults like read disturbance and retention; hence, one must provision separate BCH capability for transient faults, which we do not consider in this work.

We track the write failure probability of each simulated block to determine when the block is retired. We provide a number of spare blocks that is a parameter to simulation. We note that flash SSDs may temporarily utilize spare blocks to aid garbage collection. Because the garbage collection is orthogonal to the main idea of this work, we do not take the spare blocks used for garbage collection into consideration and assume that the provided spares are dedicated to bad block management.

3.1. Lifetime Improvement

We first look at how data dependent sparing improves the lifetime of a device. The modeled device uses BCH-20 and has 20% over-provisioning. The write failure frequency threshold is 10%, i.e., a block is retired if the probability of write failure on it reaches 10%. Fig. 3 plots our result: the percentage of surviving blocks in a storage device (Y axis) as a function of successful writes per block (X axis).

The result shows that when the number of surviving blocks with data dependent sparing is 100%, only 22% of the blocks survived with static sparing. In other words, data dependent sparing offers 78% point more physical storage capacity than static sparing before bad blocks happen. In terms of storage device lifetime—time until the first bad block occurs after consuming all spares—data dependent sparing's advantage is clear; it increases the lifetime by 18.1% compared with static sparing. Similar advantage is achieved with Flash and its result is omitted for brevity.

The two curves in the plot have a noticeably different shape. Static sparing starts to lose storage blocks soon and keeps losing more and more blocks as they become faulty. On the other hand, data dependent sparing sheds blocks much later but loses many blocks near the end of usable lifetime. *Data dependent sparing salvages otherwise bad blocks and effectively extracts more lifetime from better-than-bad blocks.*

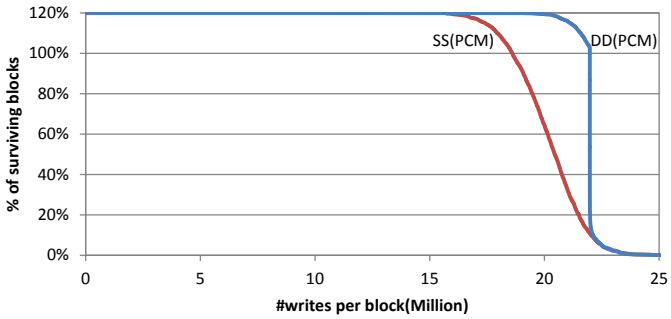

Fig. 3. Lifetime of PCM blocks with BCH-20 and 10% failure frequency threshold. "DD" denotes data dependent sparing and "SS" static sparing.

3.2. Sensitivity to Over-Provisioning

To study the sensitivity of data dependent sparing to the amount of over-provisioned capacity, we define and use a metric dubbed *lifetime increase*. It is the difference between lifetime with data dependent sparing and lifetime with static sparing, divided by lifetime with static sparing. In essence, this metric expresses the relative lifetime advantage with data dependent sparing.

We examine four different over-provisioned capacities of 1%, 5%, 10% and 20% for data dependent sparing, while keeping 20% over-provisioning for static sparing. We assume BCH-20 and a block failure frequency threshold of 10%.

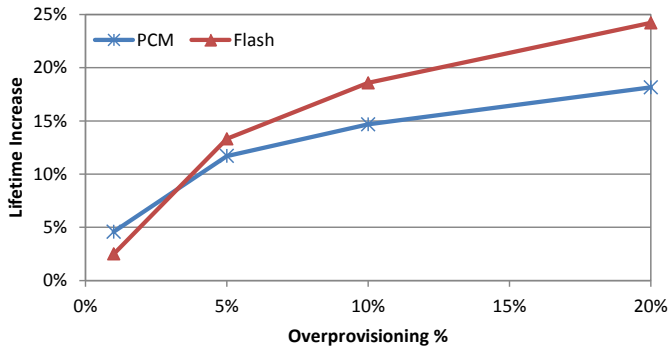

Fig. 4. Lifetime increase achieved by data dependent sparing at various levels of over-provisioning compared with static sparing with 20% over-provisioning.

Fig. 4 shows the result. Data dependent sparing is shown to outperform static sparing at all over-provisioned capacity levels examined. Data dependent sparing beats static sparing (with 20% over-provisioning) with only 1% over-provisioning and increases the lifetime by up to 4.5%. At 20% over-provisioning (i.e., same spare capacity for both schemes), lifetime increase reaches 18.1% (PCM) and 24.2% (Flash). The result proves that the observation we make in this paper that errors are typically fewer than faults in a block is extremely valuable; indeed, *data dependent sparing significantly increases the amount of data written to a block*. This explains why data dependent sparing achieves a target lifetime with a smaller over-provisioned capacity than static sparing.

3.3. Sensitivity to BCH Capability

This section studies the effect of changing the capability of the error correction mechanism. We experiment with BCH-5,

BCH-10, BCH-15 and BCH-20 to revel the effect. We assume 20% over-provisioning and a block failure frequency threshold of 10%. Our evaluation metric is lifetime increase.

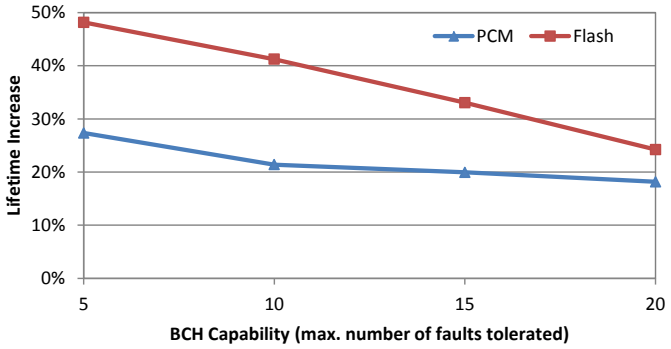

Fig. 5. Lifetime increase achieved by data dependent sparing relative to static sparing for various BCH code capabilities.

Fig. 5 plots the result. It is shown that *lifetime increase with data dependent sparing is larger when a weaker BCH code is used*. This result may look surprising at a first glance. However, with a weaker BCH code static sparing retires faulty blocks and starts to consume spares sooner. By comparison, data dependent sparing continues using better-than-bad blocks and saves wearing of spares, which translates into a significant gain in lifetime. Our result shows that data dependent sparing is *robust* in improving the lifetime of a device across the capability of an error correction mechanism used.

3.4. The Effect of Fail Frequency Threshold

In this section, we study the influence of the failure threshold upon which a block is retired. We protect the memory blocks with BCH-20 and provide 20% spares. We compare two thresholds—5% and 10%.

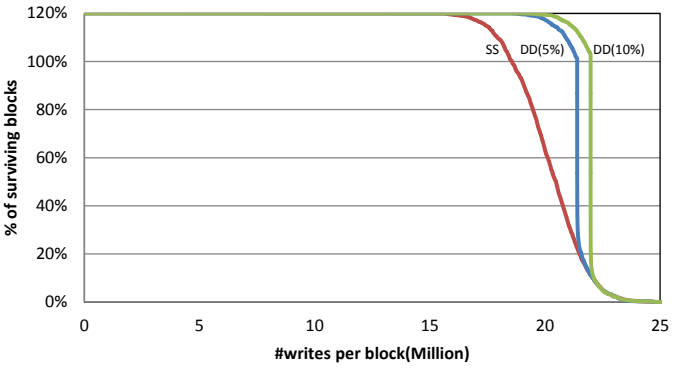

Fig. 6. Lifetime of PCM blocks under two failure frequency threshold values: 5% vs. 10%. "DD" denotes data dependent sparing and "SS" static sparing.

Fig. 6 shows the result. We find that a smaller threshold value leads to a shorter lifetime; this is expected because data dependent sparing discards a better-than-bad block more quickly with a smaller threshold value. Still, data dependent sparing under both threshold values achieves substantial gain in storage block lifetime. Our result confirms an interesting design trade-off—fail frequency threshold affects the lifetime of the storage device and management overheads.

3.5. Sparing Overhead Reduction

Finally, this section revisits the question of how much over-provisioning is needed for data dependent sparing to achieve a target lifetime, compared with static sparing. We assign static sparing with 20% and 10% over-provisioning and obtain the over-provisioning for data dependent sparing. We use BCH-20 and a failure frequency threshold of 10%.

TABLE 1
Required over-provisioning for data dependent sparing to match static sparing lifetime.

		DD Over-provisioning
20% Sparing (SS)	PCM	0.4%
	Flash	0.7%
10% Sparing (SS)	PCM	0.1%
	Flash	0.4%

Table 1 highlights the capability of data dependent sparing in reducing the amount of over-provisioning; it requires as low as 0.4% over-provisioning to achieve the same lifetime as static sparing with 20% over-provisioning and only 0.15% over-provisioning to match the lifetime of static sparing with 10% over-provisioning. Clearly, data dependent sparing has the good potential to increase the value of a storage device by achieving a target lifetime with reduced cost.

4. Concluding Remarks

We argued that existing strict bad block retirement strategies are inefficient for memories that exhibit a stuck-at fault model. A block is retired unnecessarily early when it can still be written successfully with high probably due to the data dependent nature of errors.

This paper advocates a new block retirement and sparing approach called *data dependent sparing*. By delaying the retirement of faulty yet usable storage blocks, data dependent sparing extends the lifetime of a storage device substantially, or achieves a target lifetime with much smaller sparing overheads than conventional, conservative bad block management. The added system design complexity with data dependent sparing is very small.

References

[1] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson, "Onyx: a protoype phase change memory storage array," in *HotStorage*, 2011.

[2] S. Boboila and P. Desnoyers, "Write endurance in flash drives: measurements and analysis," in *FAST*, 2010.

[3] D. K. Bose, R. C.; Ray-Chaudhuri, "On A Class of Error Correcting Binary Group Codes," *Information and Control*, vol. 3, no. 3, pp. 68–79, march 1960.

[4] L. Grupp, A. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. Siegel, and J. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," in *MICRO-42*, 2009.

[5] J.-Y. Jung, K. Ireland, J. Ouyang, B. Childers, S. Cho, R. Melhem, D. Mosse, J. Yang, Y. Zhang, and A. Camber, "Characterizing a real pcm storage device," in *NVMW*, 2011.

[6] S. Lee and T. kim, "Lifetime management of flash-based ssds using recovery-aware dynamic throttling," in *FAST*, 2012.

[7] Micron Technology, Inc, "Nand flash design and use considerations," www.micron.com, 2006.

[8] Micron Technology, Inc., "Bad block management in nand flash memory," www.micron.com, 2011.

[9] Numonyx, Inc. (Micron Technology, Inc.), "Numonyx phase change memory (p8p)," www.micron.com, 2009.

[10] C. Park, P. Talawar, D. Won, M. Jung, J. Im, S. Kim, and Y. Choi, "A high performance controller for nand flash-based solid state disk (nssd)," in *NVSMW*, 2006.

[11] M. K. Qureshi, "Pay-as-you-go: low-overhead hard-error correction for phase change memories," in *MICRO-44*, 2011.

[12] S. Schechter, G. H. Loh, K. Straus, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," *SIGARCH Comput. Archit. News*, vol. 38, June 2010.

[13] J. Yun, "Flash memory reliability model based on operations and faults," in *NVRAMOS*, 2011.